

Mobile Development Final Coursework Report

Concept development

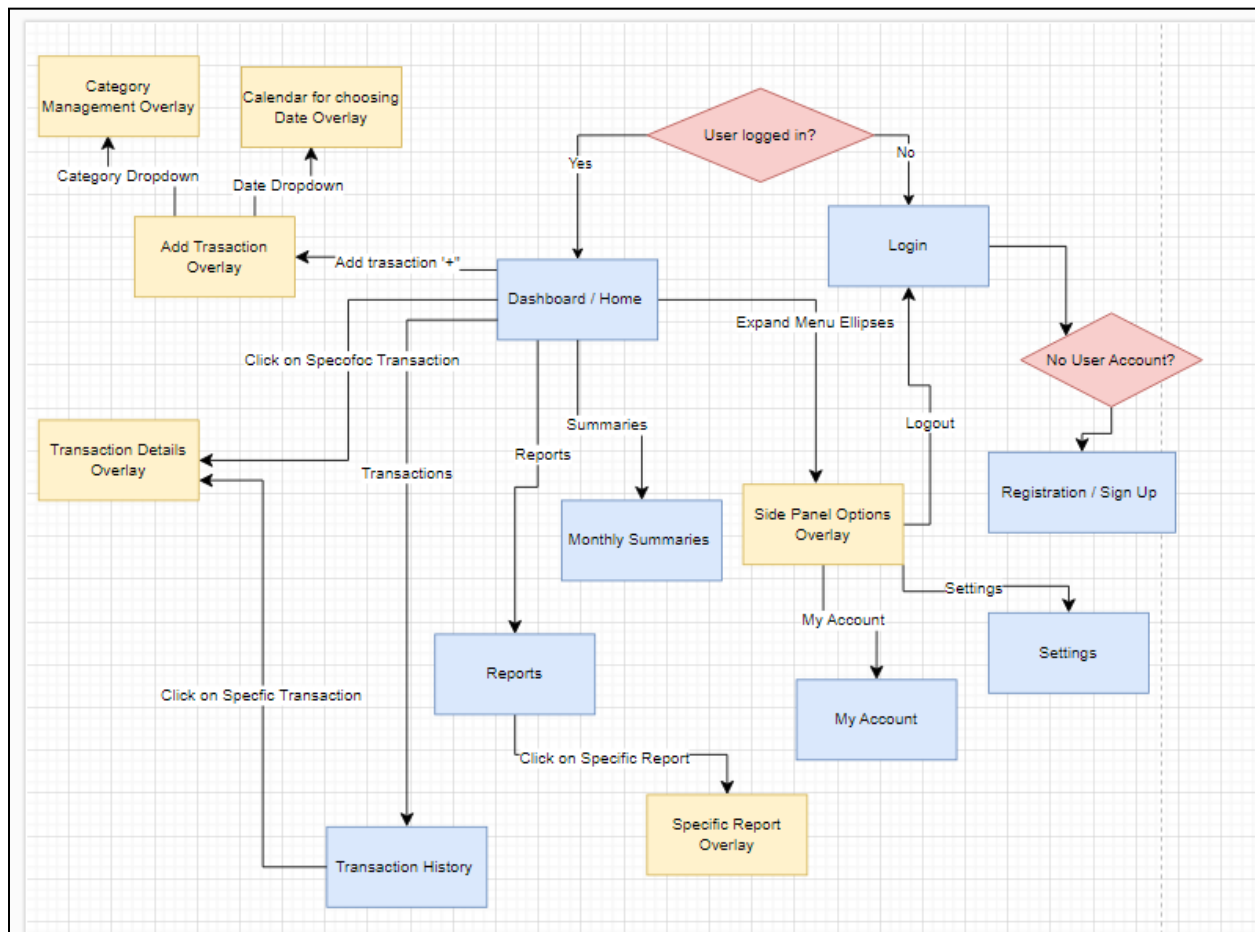
A great deal of thought was put into the planning aspect of this project. The first thing to be considered was the project ideation. The ultimate decision to create an expense tracking app was influenced by a variety of different things. One major influence has been a personal experience with apps in this domain currently on the market today. What expense tracking and similar financial oriented apps set out to provide, a noticeable marketplace gap has been identified, especially when it comes to certain focal points. Many of the apps available offer a more sophisticated focus and approach to budgeting, in particular, rather than a simplistic approach to the ease of expense tracking and providing information with respect to one's financial situation. This presented a great deal of potential for the concept of the final coursework project as the simpler focus of expense tracking speaks more to my capabilities as a learner and beginner developer. This personal inspiration along with a variety of external feedback from family and friends ultimately cemented the idea for the app.

The project planning came in a number of different forms in order to lay everything out and ensure all the necessary components were not only thought of, but possible. Possible either in terms of a practical consideration with respect to time or possible in that any components dependent on other pieces of technology were components I was equipped to provide. Learning new skills of course takes time. All in all, the project planning and ideation eventually led to a series of reasonable goals.

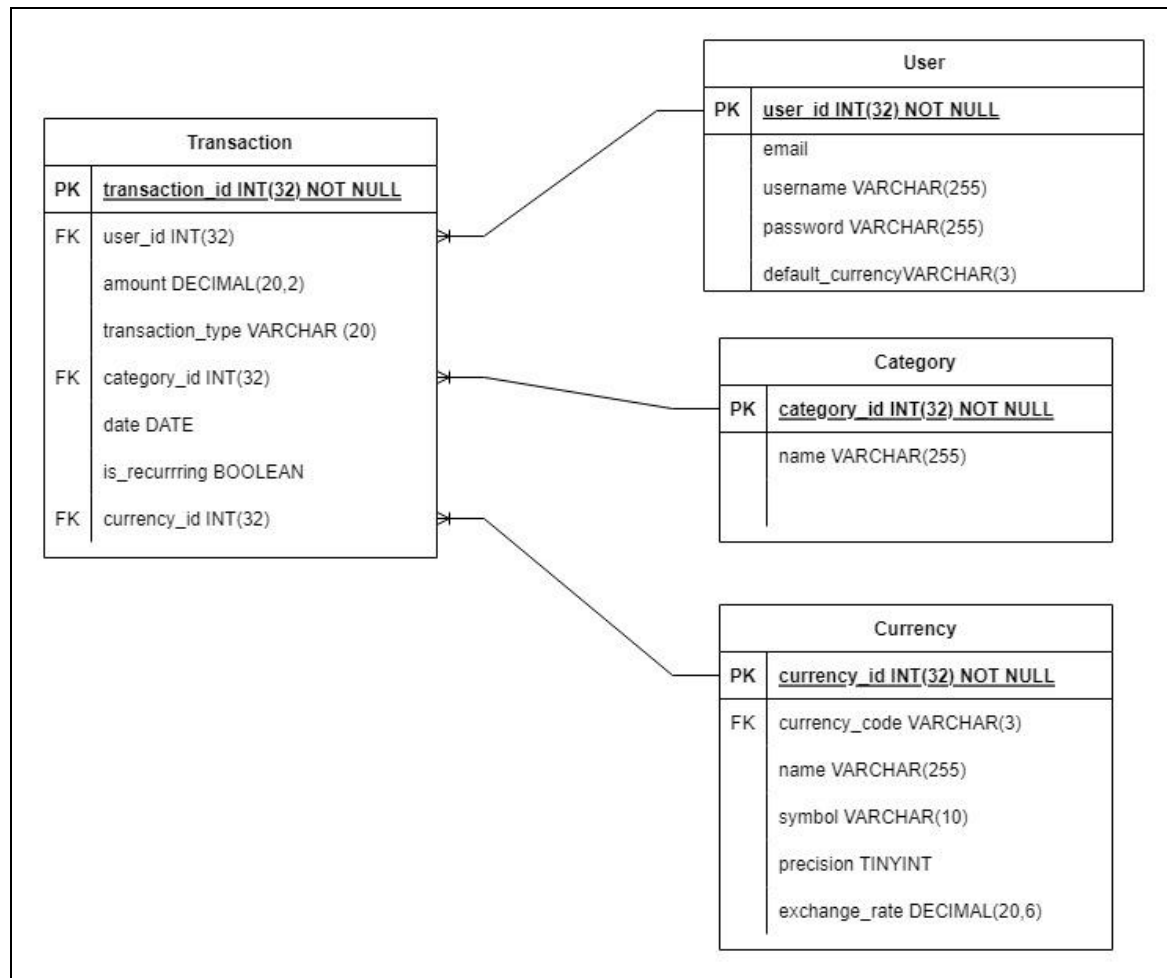
In the coming sections, further insight will be provided into the project planning, implementation, and testing of the app as it has come to fruition.

Wireframing / Prototyping / User Flow Diagram / ER Diagram

In order to begin to fine tune the general idea for the app, it seemed appropriate to begin to break it down into steps. This is where the likes of the wireframes, prototypes, user flow diagrams, and ER diagrams came into the picture. It was really beneficial to have each of these different components, and especially to be working on them together. The fine tuning of one seemed to often lead to the fine tuning of another. It would often create a spontaneous byproduct of ideas with respect to the different areas. As an idea was developed during the wireframing process, for example, I was then able to update the data models and user flow diagrams, to see how that decision fit into the design pattern overall. The planning diagrams have all gone through a series of iterations in this process. Below, all of the final iterations of these diagrams have been labelled and included:



User flow diagram



ER diagram

LOGO

DashboardSummaryReports

JULY

MONTHLY NET INCOME

Work4472.00

Cat sitting

Gifts

Other

TOTAL MONTHLY INCOME4472.00

FIXED MONTHLY EXPENSES

Rent1300.00

Broadband200.00

Gas + Electric200.00

Other400.00

TOTAL FIXED BILLS2100.00

VARIABLES MONTHLY EXPENSES

Groceries1300.00

Dining Out200.00

Socialising200.00

Other400.00

TOTAL VARIABLES EXPENSES1100.00

TOTAL MONTHLY EXPENSES

3200.00

INCOME - EXPENSES = NET SAVINGS / LOSS

3200.00

Choose Category

Default

Default

Default

Default

Default

Default

Default

Default

Create New Category

Category name

+

Choose colour

OK

Expense

Description

Sainsburys

Date

20/07/2023

Category

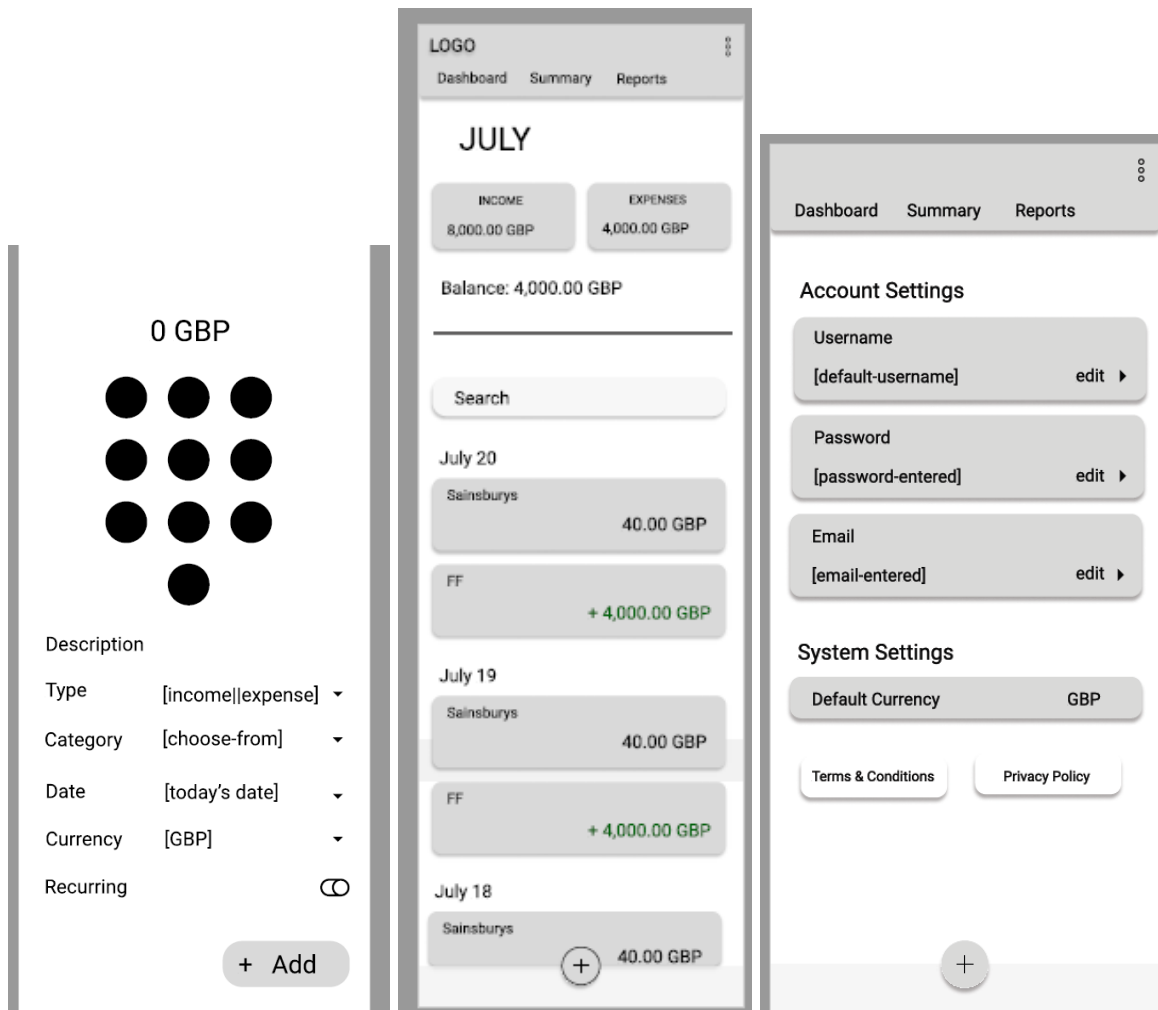
Groceries

Entered by

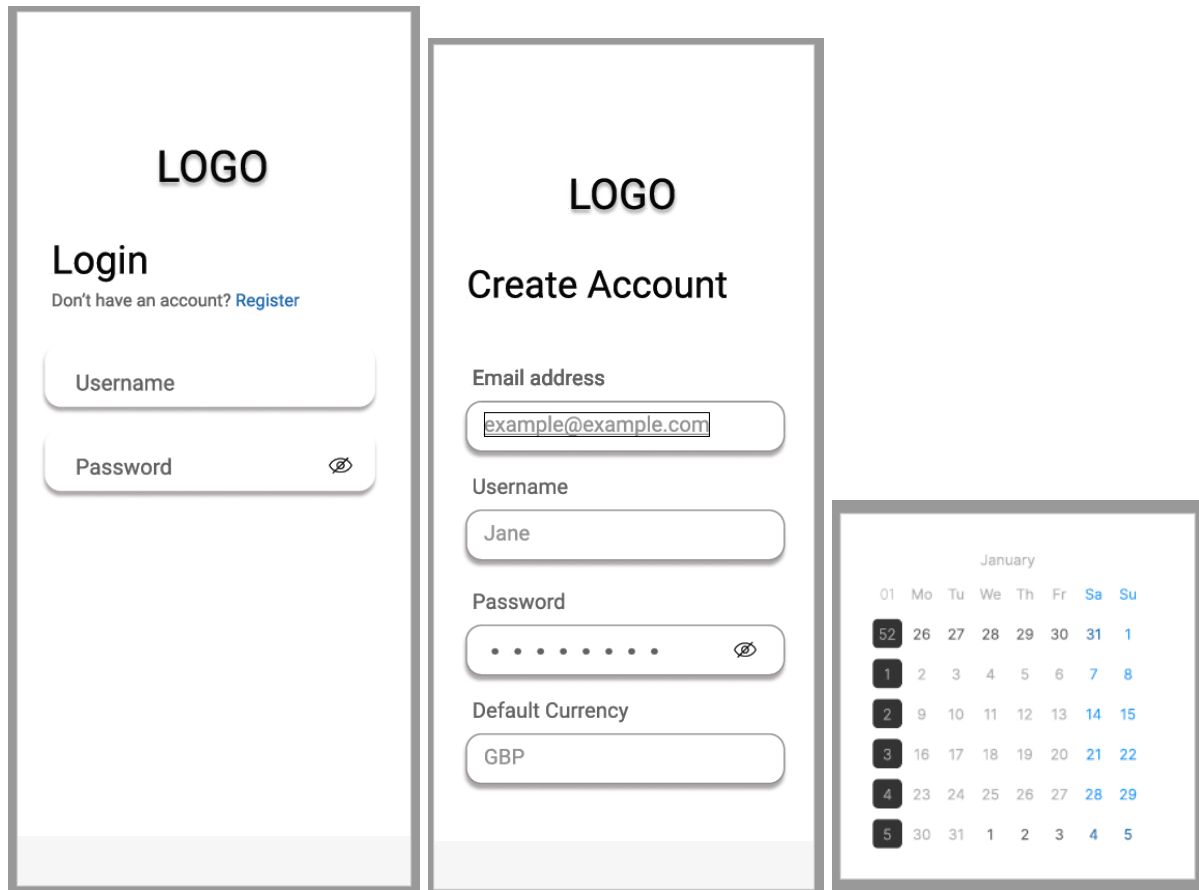
[current-user]

40.00 GBP

Wireframes - pt. 1



Wireframes - pt. 2



Wireframes - pt. 3

Another step taken in the planning process was to map out a user flow hierarchy and list of possible user actions. Using this, I was able to highlight the parts that were either a new screen, a user action, or an overlay (modal). This process has been included below and is as follows:

User Flow Hierarchy / Possible User Actions:

Blue - Screen

Red - User Action

Yellow - Overlay

User is not Logged In:

Login

- User makes Successful Login
 - Dashboard
- User clicks register
 - Registration
 - User successfully registers
 - Dashboard

User is Logged In:

Dashboard

- Menu Expansion
 - Menu Expansion Overlay
 - Settings
 - User can edit username, password, email, and default currency
 - Links to Terms of conditions and Privacy Policy
 - Logout
 - Pressing logout from the Menu Expansion takes user back to Login page
- Navigate via Top Bar Nav Links (Navigation links available on most pages - same functionality on any other page as described below)
 - Dashboard
 - Takes User to top of Dashboard Screen
 - Summary
 - User can select month from dropdown and display current monthly summaries from that month
 - Settings
 - User can edit username, password, email, and default currency
 - Links to Terms of conditions and Privacy Policy
- Press Add Transaction Button
 - Add Transaction Overlay
 - User can add transaction which includes the following information:
 - User can first enter the amount for the transaction on a keypad interface
 - Below the keypad are options to add additional information:
 - Transaction type - Choose Income or Expense from Dropdown

- Category - Pops up Overlay with list of default categories to choose from as well as option to create new, selection is added when user clicks ok and is redirected to the Transaction overlay screen with selection populated
- Date - default to today's date but can be modified by clicking dropdown, brings up calendar overlay to select date from, user is redirected to transaction overlay with date selection populated
- Currency - default to selected currency (will be first set on account registration but can be changed in settings), can be changed selecting from dropdown list
- Option to toggle whether or not the transaction is recurring - If so, Transaction will be included monthly. Recurring transactions are visible on Monthly Summary Screen
- Transaction is added when user clicks add as long as essential information is filled out
- User clicks Transaction object
 - Specific Transaction Details Overlay pops up
 - Description (Modifiable)
 - Date (Modifiable)
 - Category (Modifiable)
 - Entered by - (Non-modifiable)
- User Searches
 - Use logic to filter search but does not require a separate page

Upon completing the list of potential user actions, it was much more obvious what the required pages and components were. These have been laid out and are as follows:

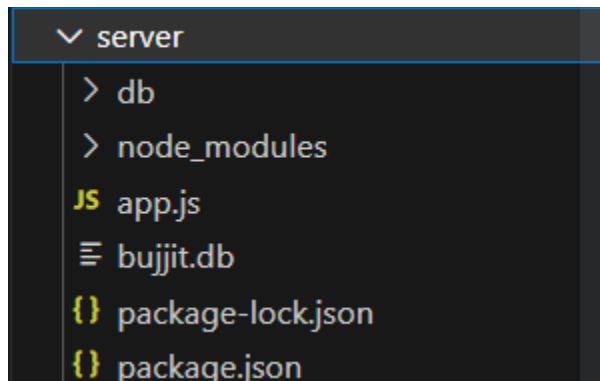
Pages:

- Login
- Dashboard
- Registration
- Summary
- Settings

Overlays:

- Main Add Transaction Overlay
- Category Management Overlay
- Date Selection Overlay
- Specific Transaction Details Overlay
- Menu Expansion Overlay

For database integration, it was decided to use Express and Node.js as the backend services for the app. These were the technologies taught in the course and the combination of back end technologies I was already most familiar with. The database chosen to implement the data models was sqlite3. All of the backend component services are contained and managed inside the server directory (a part of the same parent container directory with the app):



Project directory

With this configuration, the backend was set up to fetch data from the database and respectively deliver them to the different areas of the front end where necessary. An example of the backend code found in *app.js* handling the calls made on the backend and the communication management between the server and the front end can be found below:

```

const express = require('express');
const app = express();
// Set port for app to run on
const port = 3000;
const bodyParser = require('body-parser');

// Setup middleware for json parsing
app.use(express.json());
app.use(bodyParser.json());

// Import sqlite3
const sqlite3 = require('sqlite3').verbose();

// Define path to database file
const databasePath = './db/bujjit.db';
// Create new instance
const db = new sqlite3.Database(databasePath);

// Get endpoint for root path
app.get('/', (req, res) => {
  // Fetch first entry from transactions table

```

app.js

The database design and tables have been developed based on the different aspects of data storage the app demands. The different tables have already been identified via the ER diagram in the previous section. The tables have been laid out below with all the respective fields and data types they require:

Transactions

- id **PK**
- description (VARCHAR 255)
- user_id INT(32) **FK**
- amount DECIMAL (20,2)
- transaction_type VARCHAR(20)
- category_id INT(32) **FK**
- date DATE
- is_recurring BOOLEAN
- currency_id INT(32) **FK**

Users

- id INT(32) **PK**
- email VARCHAR(255)
- username VARCHAR(255)
- password VARCHAR(255)
- default_currency VARCHAR(3)

Categories

- id INT(32) **PK**
- name VARCHAR(255)

Currencies

- id VARCHAR(3) **PK**
- code VARCHAR(3)
- symbol VARCHAR(10)
- precision TINYINT
- exchange_rate DECIMAL(20,6)

For instantiating the tables in the database, the following sql statements are required:

Transactions

```
CREATE TABLE Transactions (  
  id INTEGER PRIMARY KEY,  
  description VARCHAR(255)  
  user_id INT(32),  
  amount DECIMAL(20, 2),  
  transaction_type VARCHAR(20),  
  category_id INT(32),  
  date DATE,  
  is_recurring BOOLEAN,  
  currency_id INT(32),  
  FOREIGN KEY (user_id) REFERENCES Users (id),  
  FOREIGN KEY (category_id) REFERENCES Categories (id),  
  FOREIGN KEY (currency_id) REFERENCES Currencies (id)  
);
```

Users

```
CREATE TABLE Users (  
  id INTEGER PRIMARY KEY,  
  email VARCHAR(255) UNIQUE,  
  username VARCHAR(255) UNIQUE,  
  password VARCHAR(255),  
  default_currency VARCHAR(3),  
  FOREIGN KEY (default_currency) REFERENCES Currencies (code)  
);
```

Categories

```
CREATE TABLE Categories (  
  id INTEGER PRIMARY KEY,  
  name VARCHAR(255)  
  color TEXT  
);
```

Currencies

```
CREATE TABLE Currencies (  
  id INTEGER PRIMARY KEY,  
  code VARCHAR(3) UNIQUE,  
  symbol VARCHAR(10),  
  precision INTEGER,  
  exchange_rate DECIMAL(20, 6)  
);
```

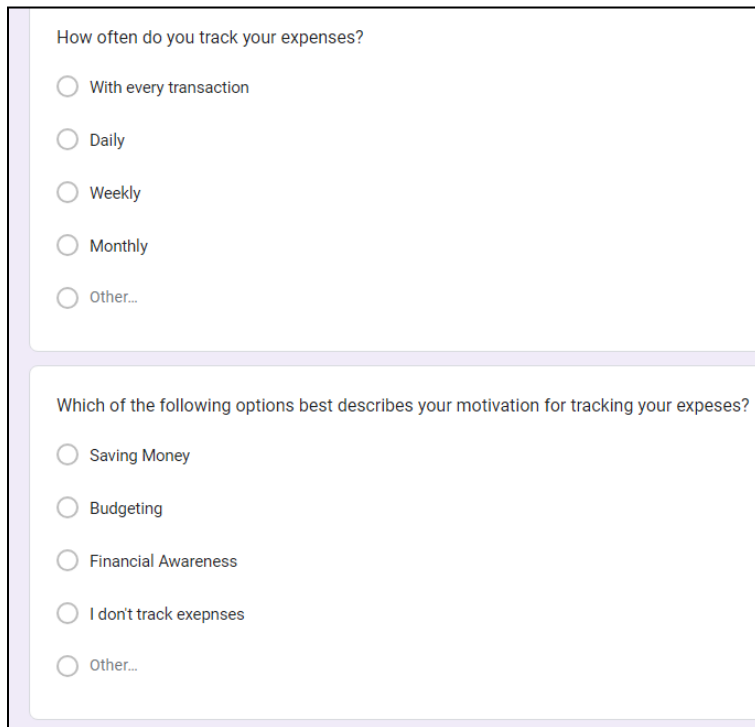
User feedback

User feedback for the app has been provided in a number of iterative stages throughout both planning and development. A great deal of the user feedback given was provided informally. I would have friends and family test the app whenever I could to gain as much feedback and outside perspective as possible. On top of the informal testing, I also sent out a survey containing the following ten questions:

1. What is your age?
2. What is your gender?
3. What is your occupation?
4. Have you ever used an expense tracking app before?
5. How often do you track your expenses?
6. Which of the following options best describes your motivation for tracking your expenses?
7. Please rate how strongly you agree or disagree with the following statement: "I am aware of the expense tracking apps available on the market today"
8. Please rate how strongly you agree or disagree with the following statement: "I am interested in using an expense tracking app".
9. If applicable, please select any combination of the following concerns or barriers that might prevent you from using an expense tracking app.

10. Of the following two options, which type of user interface do you prefer? (Clean and simple vs. Detailed and comprehensive)

This survey was created in google forms and, through its utility, many more people were made accessible in the way of receiving feedback. I received thirty-four responses which provided very valuable feedback, particularly with respect to gaining a better understanding of the use case and the things potential users were making a pattern of caring about. The formal survey was more valuable in the early stages of development, but provided a sense of direction and areas their input would be important to focus on. A few examples of the survey questions can be found below:



How often do you track your expenses?

- ☐ With every transaction
- ☐ Daily
- ☐ Weekly
- ☐ Monthly
- ☐ Other...

Which of the following options best describes your motivation for tracking your expenses?

- ☐ Saving Money
- ☐ Budgeting
- ☐ Financial Awareness
- ☐ I don't track expenses
- ☐ Other...

Screenshot of user survey - pt. 1

Please rate how strongly you agree or disagree with the following statement "I am interested in using an expense tracking app".

1 2 3 4 5

Strongly disagree Strongly agree

If applicable, please select any combination of the following concerns or barriers that might prevent you from using an expense tracking app.

- ☐ Concerns for personal privacy
- ☐ No desire to track finances
- ☐ No desire to track finances electronically
- ☐ All financial data is already stored in an alternative location (Too much work to move)
- ☐ Other...

Screenshot of user survey - pt. 2

One of the primary take-aways from the formal survey and feedback was in the amount of people that voted the option of 'financial awareness' when it comes to tracking spending. Additionally, there were significantly more votes for a user interface that was clean and simple, over one that is detailed and comprehensive. These two considerations were both potential focuses for this project, so proved to be very insightful and as a kind of reinforcement and encouragement of the idea being carried out.

On top of the formal survey, the informal feedback provided even more significant contributions to the project direction, iteratively and continually so throughout the developmental process. The informal user feedback provided vast amounts of critical feedback and information. It was often surprising how many of the comments the users made were things that, as the developer, I had ever noticed. In some cases, I probably didn't even have the ability to. This really enlightened the value of other people's perspectives and the insight they have to offer. Some of the key takeaways from the informal user feedback (again, mainly from family and friends) were the direction they informed for the following:

- Coloured background for the income and expense summaries on the dashboard
- Decreased use of colours, given many accounts of user feedback indicating it was too 'busy'
- Different label names for the summary components - what is now labelled as the 'Balance' for example, had before been 'Monthly Differential'.
- Increased size of balance total as it provides a very important piece of information

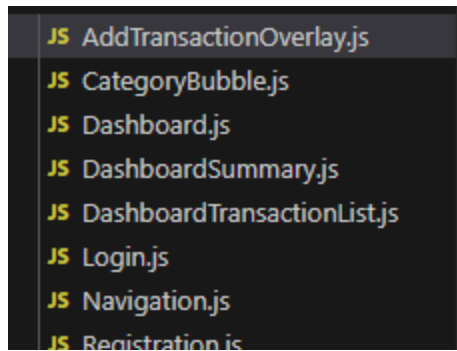
- Stylistic alterations on the search bar, containing a much thicker component border which many advised was 'clunky' looking
- Reduced amount of simplicity in the transaction list on the page. It now contains only the description and the amount, whereas before I had included further data for the transaction. Many commented it was too much information and not easy enough to 'digest'.
- As a final example, the inspiration for the category management overlay was inspired by the user feedback. I had initially designed a simple dropdown feature containing all of the categories available, but many encouraged the expansion of the category feature indicating it's an important part of the transaction information, especially with respect to the way the app is set up.

These are only a few examples of the user feedback being taken into account and iteratively guiding the direction of the app. It was easy to see how the feedback vastly improved the overall product and how invaluable this information was to receive.

Development

The development for the app was made much simpler after such careful planning. Having the pages already laid out, the required components for all the various pieces, the data models they would need to draw on, it was just a matter of implementing them. The implementation still provided many challenges, however. It was the first time developing in React Native and Expo, and of course as it is with learning anything new, there is usually a learning curve. The documentation for both proved to be a great source of support on many different occasions while running into bugs. Additional support was found in the course content, the Slack forums (chatting with classmates), and other various online forums as well. The culmination of this online support network provided huge contributions to the success of the project.

As it was laid out in the planning section, many of the components have been implemented modularly. The app took on further modular developments as the development went on. Some of the components were becoming too long-winded and complicated. In these cases, the code was broken down into simpler parts. You can see an example of this in the screenshot below, the directory provides an indication of the Dashboard component having been broken down into smaller parts - one for the summary section at the top, one for the transaction list component, etc.



Project directory

In the developmental phase, the biggest challenge was often found in discovering the best way to implement an idea that I had, in a language I was not familiar with. It also sometimes took many hours of researching to find the answer as to why a certain piece of code was not working as expected, oftentimes it was the result of a syntactical subtlety or the code producing something I was not expecting. Again, it was with the support of the several different means indicated above that contributed to sorting most of the issues and breaking down barriers. More will be discussed of the project's successes vs. shortcomings in the upcoming evaluation section.

Evaluation

Firstly, on the topic of evaluation, we will take a look at how the project has fulfilled some of the best practices for clean code and provide examples. Four areas pertinent to clean code practices are as follows:

- Inclusion of comments
- Modular code
- Having multiple files
- Consistent and description naming

Screenshots have been included to demonstrate an example of each of these in practice:

Demonstration of extensive use of comments:


```
// Function for validating form input
const validateForm = () => {
  // Set validation state to true by default
  let formIsValid = true;
  // Initialize empty object for errors
  const errors = {};

  // If no amount input
  if (!amount) {
    // Update validation state to false
    formIsValid = false;
    // Set amount errors method
    errors.amount = 'Amount is required.';
    // Ensure amount can be parsed as a number
  } else if (isNaN(parseFloat(amount))) {
    formIsValid = false;
    errors.amount = 'Amount must be a number.';
  }
  // Ensure description isn't empty (trim removes whitespace)
  if (!description.trim()) {
    formIsValid = false;
    errors.description = 'Description is required.';
  }
}
```

AddTransactionOverlay.js

As you can see in the screenshot above, there are many comments throughout, at many points providing an indication of what the code is doing. This was extremely helpful in the process of development, especially when the code was worked on, and then returned to.

Demonstration of modular code:

```
// Fetch logged in user's data from AsyncStorage
const fetchLoggedInUserData = async () => {
  try {
    const userDataJson = await AsyncStorage.getItem('userData');
    const userData = JSON.parse(userDataJson);
    setUserData(userData);
  } catch (error) {
    console.error('Error retrieving user data:', error);
  }
}

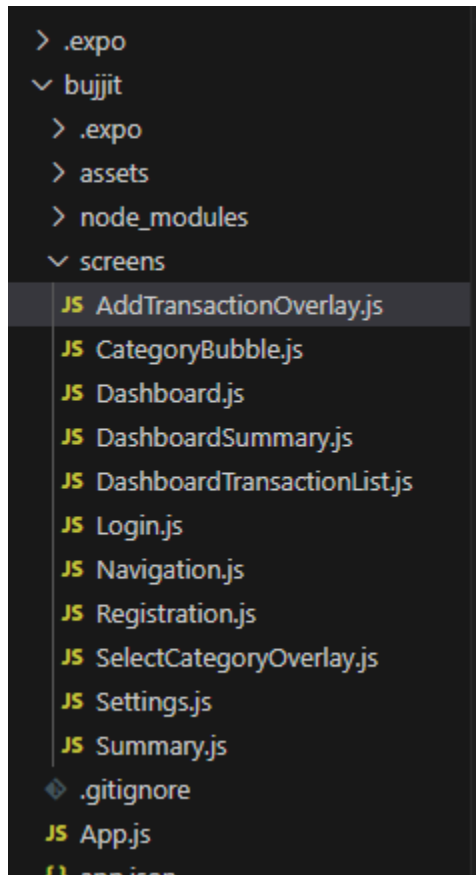
// Fetch user data from database
const fetchUser = async (user_id) => {
  try {
    const response = await fetch(`http://192.168.0.6:3000/users/${user_id}`);
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    return data;
  } catch (error) {
    console.error('Error fetching user data:', error);
    throw error;
  }
};

// Fetch category from database based on category id
const fetchCategory = (category_id) => {
  return fetch(`http://192.168.0.6:3000/categories/${category_id}`)
```

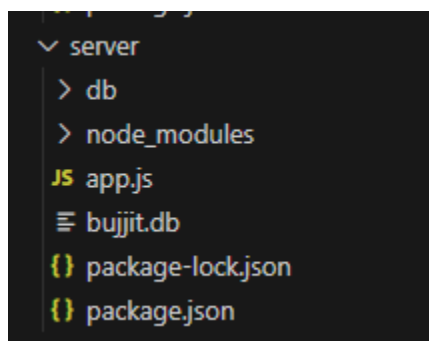
DashboardTransactionList.js

As you can see from the screenshot above, the three different functions serve different purposes, and thus have been separated according to their differing functionality, into separate functions - each one providing the functionality of fetching something different. This makes testing much easier and encourages greater integrity in the codebase when the functions operate individually (modularly) as opposed to as one enormous single threaded operation where if one thing breaks, everything does.

Demonstration of multiple file separation:



Project directory



Project directory

As you can see from the screenshot above, the functionality has been divided into multiple files according to their different contributions. Each file is named descriptively.

Demonstration of consistent and descriptive naming:

```
const handleLogin = async () => {
  try {
    // API call to the backend with input username and password
    const response = await fetch('http://192.168.0.6:3000/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        username,
        password,
      }),
    });

    const data = await response.json();
    // If the API response successful, update authentication state to true
    if (data.authenticated) {
      // Call function below to handle login success
      handleLoginSuccess(data.userData);
    } else {
      // Handle unsuccessful login
      alert('Invalid username or password');
    }
  } catch (error) {
    // Error communicating with backend
    console.error('Error logging in:', error);
  }
};

// Function to set user data when user successfully logs in
const handleLoginSuccess = async (userData) => {
  try {
    // Save userData in AsyncStorage
    // AsyncStorage.setItem('user', JSON.stringify(userData));
  } catch (error) {
    console.error('Error saving user data:', error);
  }
};
```

Login.js

As you can see from the screenshot above, all of the functions are named in a consistent naming style where words in the names are distinguished by capital letters. The names are also descriptive providing an indication of its functionality - handleLoginSuccess and handleLogin, for example, 'do what they say on the tin'.

Critical Evaluation:

Overall, I felt the project was a success. I was able to achieve the majority of the functionality I had planned for. Perhaps the greatest supporting evidence for this can be found in comparing the wireframes against the final product. The wireframes represent the idea at the beginning of the project undertaking, prior to any sort of developmental gains. It is always the challenge of development to actually bring those ideas, sometimes ambitious ones at that, to life. Though there was certainly an iterative process for this as well, and certain things that ended up needing to be adjusted, the majority of what I had planned for actually matched the ideation from the beginning quite well.

That being said, there are still many points and many areas for the project to be improved. Many of the areas in which I feel like the mark was missed was primarily due to time constraints. This was an important learning curve and an important lesson I will take moving forward. Given more time, one thing I would want to improve on is a greater coverage provided for the testing. I felt this was an area in which the project lacked as I didn't account for enough time to include it more extensively. Another aspect I would want to improve is in providing a more realistic implementation for the user authentication system. Because the app is being developed for a school project, of course it is not an actual app in production and thus the users in the database have only been added for the purposes of testing. I've used AsyncStorage for storing the user credentials. It does still authenticate the user's based on the user information found in the database, but it is simply storing placeholder text for the authentication token and not any hashed unique value. The passwords are also not being hashed. Doing this project again, I would build in the time to implement the user authentication more realistically and authentically, especially considering the real life implications beyond this course and the skills that will be required.